DINAMICA 1.0:
Multi-Stable Dynamics Analysis

Ilya Potapov

Contents

1 Introduction

DINAMICA is the tool for the automated analysis of the multi-stable dynamics. For this, it uses the differential equation tools supplied with various stochastic validation algorithms. Consequently, DINAMICA can be used as a tool for the comparative analysis between the deterministic and the corresponding stochastic systems.

Simple systems usually do not have very complex dynamics. On the other hand, as the complexity of the system grows the number of possible dynamical regimes increase, leading inevitably to the co-existing of some of the regimes. In rigid terms, for the same parameter set the system demonstrates several possible regimes and, as usual rule for the deterministic systems, different initial conditions lead to the different dynamics.

The usual assumption in physics, chemistry and biology is that the whole system consists of the equal elements of smaller size. Thus, DINAMICA considers the equation supplied by the user to be divided into sub-systems of smaller size. In principle, these sub-systems must be of the same dimension and all equal in other respects. Such a system is called a Symmetrically Coupled System.

1.1 Interface

DINAMICA has a primitive interface with no graphics carried out by the program itself. It is easy expandable for using the Gnuplot for visualization of results. The installation process can be done with or without support of the Gnuplot utility. The Gnuplot is freely available through the Internet.

DINAMICA also uses the external library (Gnu Scientific Library, GSL) to perform some basic calculations like integration of the system of differential equations, performing statistics etc. The library can be easily found and downloaded from the Web and subsequently installed.

These two are the main dependencies of DINAMICA which require the user to have them pre-installed. Although the Gnuplot is optional, the GSL is mandatory to have installed on the user's computer.

DINAMICA has beed successfully tested on Unix-like OS: Linux, Mac OS X etc. No testing was performed in the Windows environment, though the Windows use of the program is NOT prohibited. Any tests for implementation of the software on the new platforms are much appreciated and all needed help will be provided by the original author.

The most updated information regarding the interface as well as the installation and prerequisites instructions can be found in the `README` file of DINAMICA package.

1.2 Disclaimer

DINAMICS is distributed as is. The author has no responsibility for the performance of the software nor for any possible harm or damage the software might cause to the platform it is run upon. All the details of the disclaimer are explained and further clarified in the Gnu General Public License.

DINAMICA is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

DINAMICA is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

The License can be found at `http://www.gnu.org/licenses` or in `COPYING` file of the package. All the source files of DINAMICA have the Disclaimer in the beginning along with the copyright information and contact details of the original author(s).

1.3 Where To Get

DINAMICA is a part of the free-software community. The description and all source files needed for the end-user utilization and the development are located in the main software forge of the Free Software Foundation — Savannah (`http://savannah.gnu.org`). All the latest updates of the program get uploaded to the Savannah pages dedicated to DINAMICA:

`http://savannah.nongnu.org/projects/din`

No special registration is needed, the software is available right there. Various information about the package and the course of its development can be found on those pages. For example, the download area contains all public releases of the software:

`http://download.savannah.gnu.org/releases/din/`

There is also possibility to become a member of the developers' team if someone is interested in introducing the new features to DINAMICA. All such efforts are appreciated. In this regard, the package file `TODO` is a good opportunity to see all the new features and capabilities waiting to be realized.

All the changes and updates are recorded in the `ChangeLog` and `NEWS` file of the package. `ChangeLog` files are also organized by year, i.e. all updates introduced in 2012 are in `ChangeLog2012`. The most recent updates are in `ChangeLog`.

1.4 Acknowledgements

The author is thankful to all his teachers in the area of dynamical systems and stohastic processes, whom he met during the fulfillment of the Masters and Doctoral thesis. Especially, Prof. Evgenii Volkov (Lebedev Physical Institute, Moscow) and Andre Ribeiro, PhD (Tampere University of Technology).

The special thanks go to N.Devillard, who has developed the interface for using the Gnuplot utility from within a C-program
(see `http://ndevilla.free.fr/gnuplot/`). DINAMICA uses this interface.

1.5 Installation

NOTE: the most recent instructions for the installation process and all dependent procedures are located in the package `README` file.

DINAMICA uses AutoConf and AutoMake systems for configuration. The general information on how to configure the package administrated by these two systems is located in the `INSTALL` file of the package.

1.5.1 Prerequisites

1. Unix/Linux OS.
2. gcc compatible C compiler, needed for Dinamica functioning (not only compilation).
3. GNU Scientific Library (GSL) installed.
4. Gnuplot plotting utility installed (optional, but advisable).

1.5.2 Main Steps of Installation Process

1. Dowload the archive (usually .zip or .tar.gz) and uncompress it.
2. Configure the systemm by typing `./configure`. This will check for all the requirements and complain if any of those is not found. You may consider `CPPFLAGS` and `LDFLAGS` variables, as well as `--prefix` option to `./configure`, before configuring the system (see below).
3. Type "make" to compile the `libdin.a` and the `dinamica` itself. The two must appear under the `src/` directory in the root, i.e. where you uncompressed the archive. (It is also important to know why we need these two files for the software to work.)
4. Type `make install` to install `dinamica` executable and `libdin.a` library to the usual destinations (`/usr/local/bin` and `/usr/local/lib`,

respectively). This might require the root password. You may uninstall the program later by typing `make uninstall` to remove those two files from the system. After the installation one might want to remove all the files extracted from the archive.

1.5.3 Important to Know Before Configuring the Package

DINAMICA processes the input from the user (equations, parameters, variables, constants etc.) in the form of the file having `.ode` extension (ode-file, for short). The result of the processing is the output `.c` file with C language definitions and functions for the user system and the binary configuration `.bcf` file. This output `.c` file is then compiled with the DINAMICA library (`libdin.*`) generating the final executable `.din`. This executable is then invoked to read the `.bcf` file and, finally, the program fires up. Schematically this process is shown in Fig. 1.

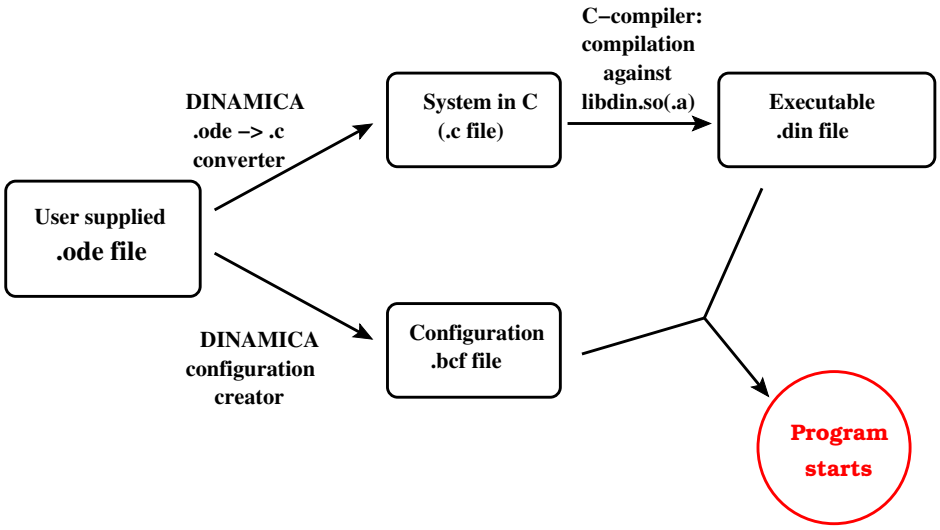

Figure 1: The general scheme representing the DINAMICA preparation procedures before it starts.

It is important to understand that the compilation and linking of the libraries take place during the functioning of DINAMICA. Thus, the compiler and the right path for the required libraries are needed to be properly set when DINAMICA is first compiled. One should take care of this before the configuration starts.

The way DINAMICA can obtain the full set of the paths is to specify `CPPFLAGS`, `LDFLAGS` and `--prefix` option, together or one by one when needed. These three variables are passed to the DINAMICA compilation command, so they are crucial. The current directory where DINAMICA is invoked is always checked for the dinamica library (`libdin.so` or `libdin.a`).

CPPFLAGS. This variable is important for the preprocessor, a program checking the included, so called header, files. During the configuration process the `./configure` script checks for several `.h` files from the GSL and the standard C libraries whether they are available. Sometimes it fails to find them in the standard locations. In this case, the `CPPFLAGS` is needed. The usage is simple: if one knows that the GSL headers are located, for example, in `/usr/local/include/`, e.g. the full path to `gsl_odeiv2.h` file is `/usr/local/include/gsl/gsl_odeiv2.h` (similarly for all other gsl `*.h` files), then one could type at the shell prompt:

```
CPPFLAGS=/usr/local/include ./configure
```

This sets the environment variable for the `./configure` script. Next, in DINAMICA this variable will be set after the `-I` flag of the compiler, i.e. will also be used as a path to the header files to find (since DINAMICA uses the same set of header files to compile the user-defined `.c` file). Note, that the path to the GSL header files is always `gsl/*.h`, so omit the `gsl/` directory when specifying the `CPPFLAGS` like in the example provided.

LDFLAGS. This variable shows the path to the GSL library (and possibly the standard C libraries). If the gsl library files are located in `/usr/local/lib` then combining with `CPPFLAGS` one could type

```
CPPFLAGS=/usr/local/include LDFLAGS=/usr/local/lib ./configure
```

which should do the trick. Additionally, DINAMICA will compile the output `.c` file against the `libdin.so/libdin.a` using this variable or the one specified by `--prefix` (see below) to find the `libdin.so/libdin.a` library. This value will go to the `-L` flag of the compiler, which specifies the path to the libraries to be used in the compilation.

- - prefix. `./configure` script accepts the `--prefix` option, which specifies the installation directory for `make install` command. Namely,

```
./configure --prefix=/usr/local
```

would install all the files produced by the package to the corresponding subdirectories of `/usr/local`. For example, binary files, i.e `dinamica`, would go into `/usr/local/bin` and libraries, i.e. libdin.a, — into `/usr/local/lib`. This variable is set after the `-L` flag to the compiler in DINAMICA.

1.6 Running DINAMICA

The most simple invocation of DINAMICA is to type at the command line:

```
dinamica <your_ode_file>.ode
```

This will fire up the program, if the configuration and installation processes went well. `<your_ode_file>.ode` is the `.ode` file containing the system to be analyzed. This file should be prepared by the user.

The program starts by showing some information about the system it has read from the `.ode` file, a little report on the compilation of the transformed `.c` file against the DINAMICA library (`libdin.so`) and some miscellaneous information. A typical output looks like:

```
<here is your prompt>$ dinamica bruss2.ode
Starting to check system's specification:
Rebuilding the function:
f(x,y)=a-(b+1)*x+y*x^2
Rebuilding the function:
g(x,y)=b*x-y*x^2
u1' = f(u1,v1)
v1' = g(u1,v1)+dv*(v2-v1)
u2' = f(u2,v2)
v2' = g(u2,v2)+dv*(v1-v2)
Starting transfer to 'bruss2.c'...
Done.
Active parameters: 'b', 'a', 'dv',
Writing 'bruss2.bcf'...Done.
Preparing for 'bruss2.c' compiling...
Compiling with: 'gcc -Wall bruss2.c
-I/opt/local/include -L/opt/local/lib -L./
-o bruss2.din -ldin -lgsl -lgslcblas -lm '
ld: warning: directory not found for option '-L/usr/oma/potapov/lib'
Starting bruss2.din...
./bruss2.din bruss2.bcf


DINAMICA Ver. 0.11 (<dl.sv.nongnu.org/releases/din/>)
Copyright 2008, 2009, 2010, 2011, 2012, 2013 Elias Potapov

Copyright 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004,
2005, 2006, 2007, 2008, 2009, 2010, 2011 The GSL Team

This software uses the gnuplot_i library written by N.Devillard
(see <http://ndevilla.free.fr/gnuplot/>).

This program comes with ABSOLUTELY NO WARRANTY;
for details type 'warranty' or simply 'w'
This is free software, and you are welcome to redistribute it
```

```
under certain conditions; see GNU General Public License for details.

Report bugs to <elias.potapov@gmail.com>


Reading 'bruss2.bcf'...Done.

>
```

Then the DINAMICA's own command line (`>`) is shown up.

DINAMICA is organized in menus, which have a certain hierarchy that does not go deeper than 2-3 levels starting from the main menu and ending at submenus. The ubiquitous commands are `ls` and `sh`/`show` (not everywhere). The former shows the list of possible menu items and the command shortcuts to access them, while the latter shows the information corresponding to a certain type of menu.

The menu items are shown such that the shortest command abbreviation for invocation of this particular menu is surrounded with parentheses. For example, menu item `(N)umerics` can be accessed by typing `n` at the DINAMICA prompt. The main menu look like:

```
>ls
MAIN menu:
(R)un*
(R)un (t)ransient*
(R)un (i)nitial*
(C)alculate*/
(F)ile/
(N)umerics/
(P)arameters*
(V)ariables*
P(e)riodics/
(G)raphics/
(I)nitials*
(T)rajectory/
C(o)ntinue/
R(a)ndom/
(Er)rors/
(S)ingularity/
(L)yapunov/
(R)un (l)inear*
```

Some more examples to clarify the concept, to access `File` menu one should type in `f` at the prompt, to run system for the transient amount of time type in `rt` etc.

The main menu also has the sign showing the type of the submenu. A slash `'/'` at the end of menu item tells a user that this is a regular menu, while an asterisk `'*'` tells about a command to be invoked.

Every command released at the prompt needs an 'Enter' key hit at the end to be accepted by the DINAMICA interpreter. One can use several commands in a row to access more items in the submenus. For example, to show the numerics information of the systems one could type in `n`, 'Enter', `sh` which will first bring the user to the `Numerics` submenu and then in that particular submenu the `show` command is invoked. Alternatively, the same result in a rather faster way can be achieved by typing `n sh` at the prompt. This is quite self-explanatory.

```
>n sh
* Dimension: 4
* Number of systems: 2
* Number of parameters: 3
* Number of user functions: 2
* Number of auxillary entities: 0
**************************
Total time: 50
Transient time: 50
Step: 0.02
Writing step: 1
Sampling frequency: 1.00
Method: rkf45
Langevin flag: false
```

2 Input Files

2.1 ODE Files

This type of files is the main source for DINAMICA. Certainly, men must tell software what to do, that is why `.ode` files preparation mostly lies upon the user's shoulders. First, we will present some general examples of using `.ode` files, which will provide one with a good basis to start his/her own research almost immediately once the examples are understood. Next, we will try to draw the detailed explanation on how the `.ode` files are constructed and what is the main syntax and usuall pitfalls one might encounter during the declaration own system.

2.1.1 Two Start-off Examples

Let's start from the simple example we used before for invocation of DINAMICA:

```
# bruss2.ode
# two Brusselator coupled by the diffusion equations
# The next statement is NOT comment and determines the number of
# coupled systems (sub-systems).
# system 2

du1/dt=f(u1,v1)
dv1/dt=g(u1,v1)+dv*(v2-v1)
du2/dt=f(u2,v2)
dv2/dt=g(u2,v2)+dv*(v1-v2)

f(x,y)=a-(b+1)*x+y*x^2
g(x,y)=b*x-y*x^2

jac u1=-(b+1)+2*v1*u1-du,u1^2,du,0
jac v1=b-2*v1*u1,-u1^2-dv,0,dv
jac u2=-(b+1)+2*v2*u2+du,u1^2,-du,0
jac v2=b-2*v2*u2,-u1^2-dv,0,dv

par b=2.5,a=1,dv=.57
init u1=10,u2=1,v1=1.5,v2=15
@ total = 50,yax=v1,yax2=v2
done
```

This file specifies the systems of two diffusively coupled Brusselators. The system of Ordinary Differential Equations (ODE's) is written down in a self-explanatory way. The variables of the system are `u1`, `v1`, `u2` and `v2`. Everthing that is not a variable in the ODE's is either parameter or function/auxillary entity. Thus, `dv` is a parameter, while `f` and `g` are the functions whose definition follows the ODE specification. The function declaration introduce new parameters: `a` and `b`. The function arguments list is within the parentheses, hence `(x,y)` denotes two arguments to the functions.

There is a possibility to include Jacobian into the system declaration. This is used by several solvers. However, it is not necessary since DINAMICA has a capability of calculating the Jacobian numerically. That is done automatically, when the program cannot find the user-supplied Jacobian. The Jacobian is included through `jac` statement followed by a variable name whose differential equation is going to be subject for differentiation. All different derivatives are separated by commas `','`. Thus, the Jacobian matrix is formed.

`par` statement specifies the initial values of the parameters of the system. If some of the parameters are omitted here, their values equal to zero by default.

`init` statement does the same job as `par`, but for the initial values of the variables. Again, omitted variable values default to zero.

`@` sign denotes the line with internal parameters for DINAMICA. In the above example, `total` means the total time of integration, `yax` denotes the variable to be plotted on the Y-axis, `yax2` denotes the second variable to be plotted along with the first one on the Y-axis.

`done` statement is not necessary, but shows the hereditary connection of DINAMICA to the Bard Ermentrout's XPPAUT software. Actually, the ODE syntax is mainly like in XPPAUT
(see `http://www.math.pitt.edu/~bard/xpp/xpp.html`).

As you might have noted the `#` sign starts the comments except for the very important DINAMICA directive `#system` which defines number of physical sub-systems that the whole system has. In the example above, this number is 2, meaning that there are 2 Brusselators coupled with each other.

This example must provide a start-off principles of defining the ODE systems through the `.ode` file.

The next example include the definition of the discrete stochastic system whose dynamics is going to be compared against the deterministic system of ODE's.

```
# Toggle Switch example
# two mutually inhibiting proteins x and y

x'=alpha/(1+y^n)-d*x
y'=alpha/(1+x^n)-d*y

init x=10,y=0
par alpha=1,n=2,d=0.1

g:alpha/(1+x^n);+y
g:alpha/(1+y^n);+x
g:d*x;-x
g:d*y;-y

@method=complex,method2=rkf45,sf=1,total=10000
done
```

This system has tow ODE's describing the dynamics of two protein species in a Genetic Toggle Switch. This systems is characterized with the two stable states and possibility to switch between them. Here you can find another type of variables/ODE definition — through `x'` notation. This totally equals the `dx/dt` notation.

The main difference as compared to the first example in this section is the `'g:'`s statements closer to the end of the file. These statements define the Gillespie procedure for solving systems possessing the discrete and

stochastic dynamics. The syntax goes as follows: `g` is a keyword, everything between `':'` and the following `';'` is known as the propensity of the chemical reaction and, finally, everything after `';'` is the update vector, i.e. the vector (whose elements are separated by commas `','`) containing the information on how the numbers of the species involved in the reaction change after the reaction takes place. In our example, first reaction produces (`+y`) one molecule `y`, while the third one removes (`-x`) one `x` molecule from the reaction space.

Under the section of internal parameters (after `@`) you can find `method=complex` directive which tells DINAMICA to use both stochastic discrete and normal ODE integration methods altogether and compare the results. Additionally, `sf=1` statement tells DINAMICA to use `sampling frequency` for the stochastic simulation equal to 1 (depending on the units used in the system, it could be seconds, years or ages).

Finally, `done` finishes the input.

2.1.2 The Syntax in Details

3 Methods

3.1 Simulation Methods

3.2 Dynamics Analysis Methods

For the analysis of the dynamics DINAMICA uses the simulated time series. At the moment, all methodology described in this section refers only to the deterministic time series that is obtained from the simulation of a system described in terms of Ordinary Differential Equations (ODE's).

First, DINAMICA analyzes the dynamics of 1D system: either the whole system, if the system's physical dimension is equal to 1, or every sub-system of the larger system (number of physical systems, or sub-systems, is determined by the `#system` directive in the `.ode` file). Next, the whole system analysis is fulfilled producing the overall dynamics report.

The whole analysis is based upon the concept of *slope*. The next section explains in detail the concept. Then, we present the slope algorithm for 1D systems with the dynamical regimes the algorithm is capable to determine.

3.2.1 The Concept of a Slope

Once the deterministic time series is simulated it can be analyzed through the DINAMICA `TRAJECTORY`-system (or T-system). The elementary unit the whole analysis relies upon is *slope* . The slope is a part of the calculated trajectory (or the time series) from the beginning of section, where the trajectory's points start to decline over time, up to the beginning of section,

where the points start to ascend over time, or vice versa. The concept is illustrated in Fig. 2.

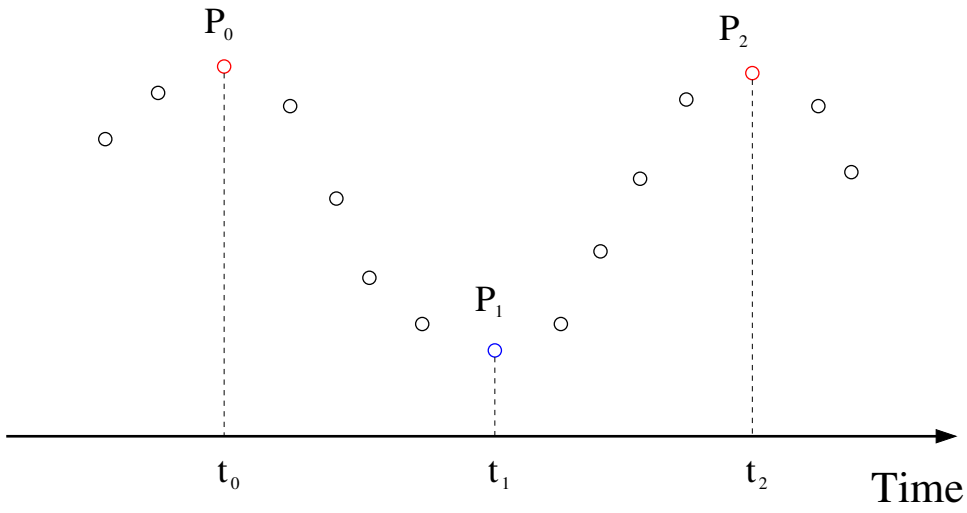

Figure 2: The graphical representation of the slope concept

The set of calculated points belonging to the trajectory has *turning points* at which the trajectory changes its direction pointing either upwards or downwards. Strictly speaking in terms of continuous trajectory and infinitesimal time step, at those points the corresponding derivative becomes equal zero.

In Fig. 2 these points are depicted in red and blue (P_0, P_1 and P_2) demarcating two slopes: the first contains all the points from point P_0 up to P_1 inclusively and the second — from P_1 to P_2.

Practically speaking, all kinds of trajectories contain the slopes, since in the most cases the calculated points have at least small discrepancies in their values caused by the computer representation of the real numbers. Nevertheless, in those really rare cases, when two adjacent points have values that are the same (up to the all representation bits of the numbers), the T-system determines the *plateau*, and the beginning and/or end of the slope is calculated as the average point between the plateau points (in case of non-integral value the ceiling operation result is taken).

Thus, one can distinguish *ascending* and *descending* slopes. In Fig. 2 the first slope is descending (from P_0 to P_1), while the second one (from P_1 to P_2) is ascending. For the obvious reasons, the ascending and descending slopes alternate over time. Additionally, the end of each slope is the beginning of the next one. For this reason, for example, the T-system of DINAMICA stores only the time that the slope starts at. Using the example depicted in Fig. 2, only t_0 and t_1 are stored for the two slopes shown.

There are three entities that describe a slope in DINAMICA. The slope

amplitude is the difference between the variable values at the beginning and at the end of the slope. The slope *base* is the lowest points of the slope. The beginning of the slope in time is the third entity.

Continuing with the example of Fig. 2, the slope appearing earlier in time has amplitude equal to $(P_0 - P_1)$, base — P_1 and the start time — t_0, while the later appearing slope has amplitude equal to $(P_2 - P_1)$, base — P_1 and the start time — t_1. As a result, two adjacent slopes (descending and then ascending) have the common base.

3.2.2 1D Dynamical Analysis

DINAMICA processes the simulated time series of a single variable within the first physical dimension of the system. All dynamics detection is based on the slope analysis of the simulated trajectory. The process could be virtually divided into the *preparation* stage and actual slope algorithm.

The preparation process starts with the calculation of the peaks and troughs of the simulated trajectory (in Fig. 2 points P_0 and P_2 are peaks and point P_1 is a trough). Then, based on the peak and trough information the slopes of the trajectory are calculated. Next, the slopes are checked to represent real slopes of the trajectory and not small "fluctuations" of it. These small fluctuations appear in systems with very different timescales, e.g. when there are very slow and very fast equations in the system. The "fluctuations" appear due to this difference of the time scales and *not* due to any stochastic forces applied to the system. Perhaps, a suitable integration algorithm accounting for the stiffnes of the system might resolve this problem. In any case, this "sanity" checking of the slopes does not hamper the whole analysis and, hence, is included in the preliminary preparation stage of the process.

Before moving to the section explaining the slope algorithm we need to take a look at the dynamical regimes and the definitions DINAMICA utilizes in the dynamics processing.

Dynamical Regimes of 1D Systems. 1D (here we refer to the physical dimension) systems can in principle have two types of dynamics: *stationary* and *non-stationary* behavior. The former refers to so called steady state of a dynamical system, at which the system eventually ends up at a stationary point. The latter, oppositely, connotes a behavior that does not rest at any particular point. The non-stationary behavior contains many sub-classes of behaviors, ranging from pure oscillatory dynamics with a distinguished period and amplitude to chaos with no certainty in the period and amplitude.

NOTE #1: the DINAMICA slope algorithm determines the *genuine* dynamics of a system in a sense of the realized (really appearing) behavior. For example, there is no way in the slope algorithm to distinguish between oscillations induced by the harmonic oscillator and the limit cycle oscillations

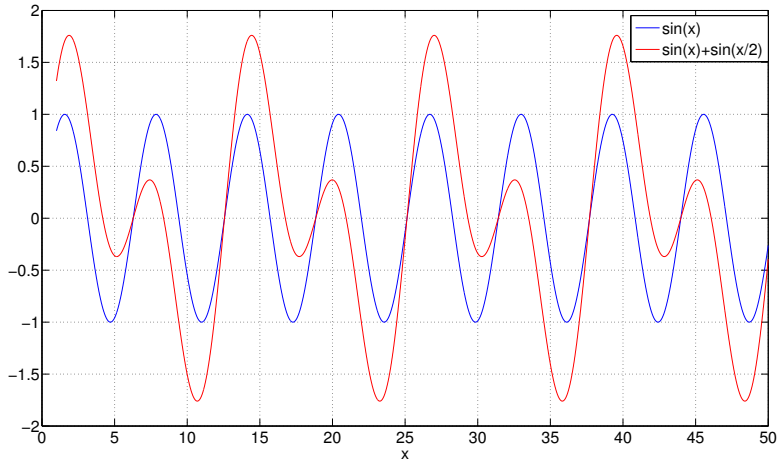

Figure 3: Sinusoidal oscillations as an example of period-1 (blue) and period-2 (red) oscillatory dynamics.

(the first one is the linear system possessing the "center" stationary point with pure imaginary Lyapunov numbers, while the second is the non-linear system containing the Hopf bifurcation).

NOTE #2: DINAMICA is capable of determining the chaotic behavior — a dynamical regime in which the deterministic laws of motion produce unpredictable behavior. But this capability of the program goes beyond the slope algorithm, which most likely would produce the "unknown" or period-n oscillatory regime (see below), where n is very large, in the case of chaos.

Thus, the non-stationary dynamical behavior is either oscillations with some predictable periodicity or chaos. The oscillatory regime can be characterized with its amplitude and period. While the amplitude value is not qualitatively useful, the period might be composed of several sub-periods, denoting, if they are present, the additional frequency of oscillations. This takes place, for instance, in case of the torus attractor.

Here, we define *period-n* oscillations referring to the oscillatory trajectory with every oscillation being of the same amplitude and the same level as those of the n-th oscillation *prior* or *next* to the given one. For example, Fig. 3 shows two sinusoidal oscillation trends. The first one is described by the expression $\sin(x)$ and the second one — by $\sin(x)+\sin(x/2)$. The former has one frequency of oscillations and the latter one has a period composed of two sub-periods. The $\sin(x)$ oscillatory regime is period-1 oscillations, the $\sin(x) + \sin(x/2)$ oscillatory regime is period-2 oscillations.

The summary of all dynamical regimes detectable within the frame of the slope alglorithm goes in Tabl. 1. Here we also present the numerical codes for each possible regimes: 0 means stationary dynamics, $1 \ldots n$ means

period-1...period-n oscillatory regime and -1 stands for the unknown or undetermined regime.

Regime	**Name**(**numerical code**)
Stationary state	SS(0)
Period-n oscillations	OS(n)
Unknown/undetermined	–(-1)

Table 1: 1D dynamical regimes detected by the slope algorithm.

The slope algorithm. Note that the dynamics of a system cannot be stationary and non-stationary at the same time. Thus, these two pathways of analysis are separated from each other in the slope algorithm (SA).

The `stationary` dynamics can be easily detected when the system is already at the rest state, meaning it does not deviated from the state significantly. This can be checked by comparing the very last slope's amplitude to the system-wide absolute error level (which is defined by the user). If the amplitude is so small (less than the error level), then the decision is made right away — `SS(0)` (see Tabl. 1).

The more complex situation of the stationary dynamics detection takes place, when the algorithm receives the time series where the system does not appear to be in its final resting state. But, since the system has it, it must converge to it. The convergence is manifested in the consistent decrease in the slope amplitudes when moving from the first slopes to the last ones. So the slopes successively demonstrate a decrease in their amplitudes as time goes on (or any other independent variable).

An example of the stationary dynamics is shown in Fig. 4, where we have taken the Brusselator system[1] with parameter set: $A = 1$ and $B = 1.9$.

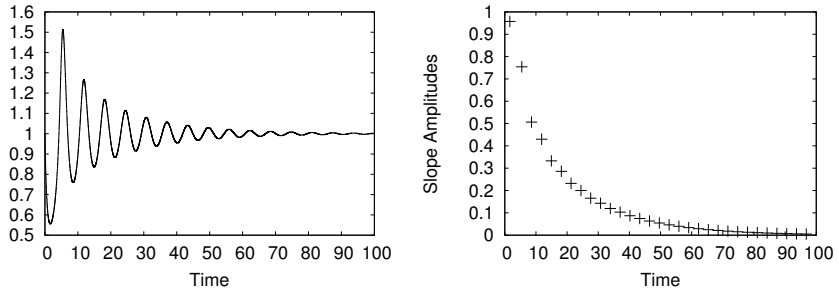

Figure 4: The stationary dynamics of one of the variables of the Brusselator (left) and the corresponding slope amplitudes (right) over time. The parameter set for the Brusselator equations: $A = 1$ and $B = 1.9$

[1] http://en.wikipedia.org/wiki/Brusselator

4 Examples